R Multivariate Regression Sandbox (Caution: Typos are Legion)

Code

This is an R Studio Markdown (http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

When in our school's Math Statistics courses, R is a typical platform that will be used in the classroom.

R is open source and can be aquired here.

https://www.r-project.org (https://www.r-project.org)

Since R is open source and is commonly used by a broad community it has gotten a large following. Also since it's open source there are a HUGE number of discipline-specific packages. (Many of them don't necessarly do statistics but since R is free…)

A common workbench end that is used with R is R-Studio which also has an open-source version that you can download from here.

This demonstrates using R to preform a multivariate regression on the Concrete Dataset

https://www.rstudio.com (https://www.rstudio.com)

R Studio also has support to bring in the many libraries that are available.

This particular example leverages this dataset.

http://kyrill.ias.sdsmt.edu/cee_284/L20_Non_Linear_Regression_Sandbox.xlsx (http://kyrill.ias.sdsmt.edu/cee_284/L20_Non_Linear_Regression_Sandbox.xlsx)

You can download it to your machine and make a small edit further down to access it with R.

You will need the following packages and their dependencies whcih can be installed with R Studio before you start.

XLConnect (for reading in the spreadsheet data), MASS (for the linear refgressions) and plyr (for satisfying your prof's need to go OCD on the original dataset)

Warning XLConnect will run off of Java and is therefore often a memory hog and a little pokey for large files but it gives you more control than other Excel R tools

This segment of the R code allows you to access these packages.

(Notice the # character. In R, the # denotes the start of a comment and can be on the same line as your code)

Hide

Hide

```
library(XLConnect) # Load the "Excel Connector for R" Library. (to load the spreadshe
et data)
library(MASS)      # Load the "Modern Applied Statistics with S Library"" (for regres
sions)
library(plyr)      # Load the "Tools for Splitting, Applying and Combining Data" Libr
ary
```

You will need to change the path to your file location. Edit the line below to accomodate where you have placed that downloaded spreadsheet,

http://kyrill.ias.sdsmt.edu/cee_284/L20_Non_Linear_Regression_Sandbox.xlsx (http://kyrill.ias.sdsmt.edu/cee_284/L20_Non_Linear_Regression_Sandbox.xlsx)

Hide

Hide

```
excel_file_name <- "/Users/wjc/Downloads/L20_Non_Linear_Regression_Sandbox.xlsx"
```

Import the data from the spreadsheet into a single data “frame” which works like a table or ledger of data.

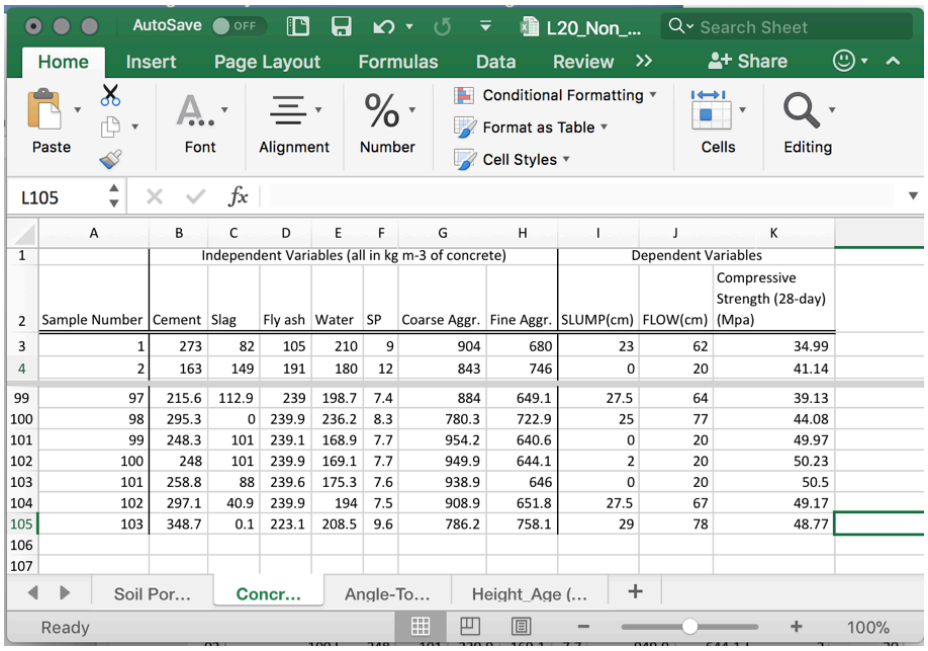
This command uses the “readWorksheetFromFile()” function which is loaded with the “XLConnect” package.

(Also when I use functions in R, I tend to get very anal and try to comment the arguments as best as I can since I don’t use R on a daily basis.)

In this case we are using a spreadsheet with multiple pages.

Our concrete data is on the sheet called “Concrete.”

We are pulling data in from a specific region of the sheet (including a header line) which goes from A2 to K105.



Screenshot of the Excel Sheet.

Here we will grab data from the sheet and call the whole group of data (the data “frame”) “exceldata”

Hide

Hide

```
exceldata <- readWorksheetFromFile(file   = excel_file_name, # file name here
                                   sheet  = "Concrete",      # select the Concrete Sh
eet
                                   region = "A2:K105",       # chose the cells you ne
ed to import
                                   header = TRUE             # TRUE if the first impo
rted row is the Header
                                   )
```

Now with the data loaded let's take a look at the inventory of the data we just imported.

Hide

Hide

```
str(object = exceldata)
```

In looking at this output, we see that the labels (or headers) are not the same as our spreadsheet.

Fix that we can fix that… with the rename function (which is in the plyr package)

(And we can make a few more tweaks along the way)

Hide

Hide

```
exceldata <- rename(x       = exceldata,              # data frame you want to patch
                    replace = c("SP" =                # the name you want to change
                                "Superplasticizer")  # the replacement name
                    )

exceldata <- rename(x       = exceldata,
                    replace = c("Fine.Aggr." =
                                "Fine.Aggregates")
                    )

exceldata <- rename(x       = exceldata,
                    replace = c("Coarse.Aggr." =
                                "Coarse.Aggregates")
                    )

exceldata <- rename(x       = exceldata,
                    replace = c("SLUMP.cm." =
                                "SLUMP")
                    )

exceldata <- rename(x       = exceldata,
                    replace = c("FLOW.cm." =
                                "FLOW")
                    )

exceldata <- rename(x       = exceldata,
                    replace = c("Compressive.Strength..28.day...Mpa." =
                                "Compressive.Strength")
                    )
```

OK now let's look at what we just fixed…

Hide

Hide

```
str(object = exceldata)
```

(That's better!)

Now onward!

Let's start by doing a "simple"" plot. In this case since I already know the answer because the spreadsheet also has a table of how well our independent variables correlate against the dependent variables (Slump, Flow & Strength). The Cement correlates the best against Compressive Strength (OK, truth be told, it correlates the least badly).

We can acutally do this with a correlate function…

To grab a value in the table “exceldata” we call the data frame (exceldata) and the variable name (Cement or Water vs Compressive.Strength), separating the frame and variable names by a $ sign.

Hide

Hide

```
print(" Cement vs Compressive Strength Correlation")

cor(x = exceldata$Cement,                # the x-value
    y = exceldata$Compressive.Strength, # the y-value
    method = "pearson"                   # method of correlation
    )
```

or if you like to do everything at once… (Not always the best thing to do)

Hide

Hide

```
print("Correlation of All variables against Compressive Strength")

cor(x = exceldata,                       # this time X is the whole data frame...
    y = exceldata$Compressive.Strength, # the y-value
    use = "everything", # correlate evything
    method = "pearson"
    )
```

Now to plot the Cement vs Strength.

First the Concrete vs Strength..

Hide

Hide

```
# Now we can plot

plot(x    = exceldata$Cement,                              # x-values (the $ lets us reac
h into
     y    = exceldata$Compressive.Strength,                # y-values     the data frame)
     main = "Example of a Single Variable Regression", # main title string
     xlab = "Cement (kg m-3 concrete)",                  # x-labels
     ylab = "28-dy Compressive Stren. (MPa)"             # y-labels
     )
```

Not so good a correlation there…

But let’s move on and create a regression model from this.

Here we will use the lm (linear model) function from the MASS package.

For the regression formula

Strength(Cement) = a0 + a1 * Cement

the “prototype” (formula) for the function is…

“Y ~ X” (with the y-intercept implicit in the formula… you don’t put it in but it’ll be there when you’re done.)

Hide

Hide

```
linear_model.S_v_c <- lm(formula = Compressive.Strength ~ Cement, # your formula y ~
x
                         data    = exceldata)                     # the data frame
```

Let’s see what we have… This summary command will provide the details of the lm() function’s important results

For us we want to see the Y-Intercept [the (Intercept) under “Estimate”] and the slope that goes with our independent value (“Concrete” under “Estimate”)

The Standard Error of the Estimate is there (Residual Standard Error) as is the Coefficient of Determination (Multiple R-squared)

We’ll talk about a few of the other features when we do the larger multivariate regression

Hide

Hide

```
summary(object = linear_model.S_v_c)
```

Now let’s make a pair of 95% confidence limits

Create a simple linear array to stretch from our min to max dependent values

I tend to overcompensate here and make a BIG array with the sequence (seq) function

Here I am making a variable called “newx” that will represent 501 values of Cement going from 0 to 500 kg m-3.

Hide

Hide

```
newx <- seq(from       =   0,  # start for a sequence
            to         = 500,  # end for a sequence
            length.out = 501)  # number of elements in a sequence
```

And here we create an array of the confidence limits using this “newx” field

(Note that use can use this function “predict” to do both prediction and confidence limits and also remember that prediction limits should ONLY be used when your original data samples conform to a normal distribution – which often isn’t the case.)

Hide

Hide

```
conf.S_v_c <- predict(object   = linear_model.S_v_c,     # your regression model
                      interval = c("confidence"),        # confidence or prediction i
ntervals
                      level    = 0.95,                   # 1-alpha (here it's a 95% C
I)
                      newdata  = data.frame(Cement=newx) # swapping out of new indep
variables
                      )
```

The resulting array, conf, will have three columns: the original linear regression using our newx as the independent value and the + & - CI’s

Hide

Hide

```
print(conf.S_v_c)
```

So let’s replot our relationship…

…our regression

and finally our lower than upper confidence limits…

(indexing on those confidence limits in “conf.S_v_c”" start at zero, by the way… so the counter is index 0, the regression line value is index 1, and the CI values are 2 & 3)

Hide

Hide

```
# Now we can plot

plot(x    = exceldata$Cement,                          # x-values (the $ lets us reac
h into
     y    = exceldata$Compressive.Strength,            # y-values     the data frame)
     main = "Example of a Single Variable Regression", # main title string
     xlab = "Cement (kg m-3 concrete)",                # x-labels
     ylab = "28-dy Compressive Stren. (MPa)"           # y-labels
     )

# And here we can plot the regression line

abline(reg = linear_model.S_v_c,  # put the regression model information here
       col = "red"                # color it red
       )

# And here we create an array of the confidence limits using this "newx" field

lines(x   = newx,           # the x data
      y   = conf.S_v_c[,2], # the y data (the low-end confidence limit)
      col = "blue",         # make the line blue
      lty = 2)              # use a dashed line

lines(x   = newx,           # the x data
      y   = conf.S_v_c[,3], # the y data (the high-end confidence limit)
      col = "blue",         # make the line blue
      lty = 2)              # use a dashed line
```

Looks pretty good! (well the graph does.. not necessarily the quality of the regression)

And now we're going to do something about that!

We're now going to use not just one independent variable… but all 7 of them!

The good news is that it follows the same form as the simple linear regression. This time we string along all of our independent variables with in our formula prototype.

Hide

Hide

```
linear_model.S_v_all <- lm(data    = exceldata,                          # your data fra
me
                           formula = Compressive.Strength ~ Cement +  # your formula
                                                            Slag +
                                                            Fly.ash +
                                                            Water +
                                                            Superplasticizer +
                                                            Fine.Aggregates +
                                                            Coarse.Aggregates
                           )
```

And here are these results…

Hide

Hide

```
summary(object = linear_model.S_v_all)
```

Our regression coefficients are still here under the “Estimate” column as are our Standard Error of our Estimate and our Coeff of Determination.

Also we can now take a good look at those asterisks at the end of line with the parameter coefficients. These can explain which independent variables do the heaviest lifting in our regression. The more asterisks, the more important the dependent variable is to the larger multivariate regression. Here, we can see that the Cement and Water are doing most of the “work” in fitting our suite of independent variables to our dependent variable of Compressive Strength.

Finally there is the P parameter for which the smaller it is, the better we can say that the relationship that we’ve made with our regression represents our dependent variable.

Now… on to looking at our results.

Here is where viewing the results of the regression is tricky.

We have 7 independent variables but we’d like to see the impact of the fit if all 7 variables on our strength

When I do this I like to plot the true y value against my regression y(x1,x2,x3,..)

So to do this I will take the fitted values of y and plot them against the original values of y

Getting the fitted values is easy.

I’m using the fitted function but you can also use the predict function from earlier with the fitted function.

Hide

Hide

```
fitted.S_v_all <- fitted(object = linear_model.S_v_all)
```

The other thing is to now do a second regression (ok a third in this entire demo…)

Here we will just regress y(x1,x2…) against y.

Hide

Hide

```
linear_model.S_v_Sofall <- lm(formula = fitted.S_v_all ~ exceldata$Compressive.Streng
th)

summary(object = linear_model.S_v_Sofall)
```

And you'll see that they have the same correlation coefficient as the earlier multivariate regression

And now, let's close this exercize and plot up our multivariate regression.

Hide

Hide

```
# first we need to set a specific graphics parameter to set our plot shape.
#   "s" makes a square, "m" is the default which maximizes the plot region.

par(pty = "s") # this makes the plot square
               # (I like square plots when I plot "apples against apples")

# now a simple x-y scatterplot as before but with both axes having
#   the same range...

plot(x    = exceldata$Compressive.Strength,                          # x-values
     y    = fitted.S_v_all,                                         # y-values
     main = "Example of a Multiple Variable Regression",            # title string
     xlab = "Obs 28-dy Compressive Stren. (MPa)",                   # x-label
     ylab = "Pred 28-dy Compressive Stren. (MPa)",                  # y-label
     xlim = c(min(exceldata$Compressive.Strength,fitted.S_v_all),  # x-axis range
              max(exceldata$Compressive.Strength,fitted.S_v_all)),
     ylim = c(min(exceldata$Compressive.Strength,fitted.S_v_all),  # y-axis range
              max(exceldata$Compressive.Strength,fitted.S_v_all)),
     asp = 1                                                        # aspect ratio bet
ween
     )                                                              #    x and y scale
s

# plot the linear regression line

abline(reg = linear_model.S_v_Sofall,  # put the regression output here
       col = "red"                     # color it red
       )

# and we can also plot a simple one:to:one line.

abline(a   = 0,     # y-intercept
       b   = 1,     # slope
       col = "grey" # color it grey
       )
```

And here we have a nice plot showing our true vs predicted values.

While here, we can do some general error metrics that may be useful..

First, the Bias… (if we are too high or too low)

Hide

Hide

```
bias = sqrt(mean(fitted.S_v_all)-mean(exceldata$Compressive.Strength))

print("BIAS")
print(bias)
```

The root mean squared error (RMSE) thought he standard error of the estmate is technically the one we use here. RMSE remains a common error metric though…

Hide

Hide

```
rmse = sqrt(mean( (fitted.S_v_all - exceldata$Compressive.Strength)^2)  )

print("RMSE")
print(rmse)
```

And finally our correlation coefficient (which is basically our coefficient of determination before the “R” is “squared”)

Hide

Hide

```
r = cor(x = fitted.S_v_all,                  # the x-value
        y = exceldata$Compressive.Strength, # the y-value
        method = "pearson"                   # method of correlation
        )

print("correlation coefficient")
print(r)
print("coefficient of determination")
print(r^2)
```

And with that, we’re done.